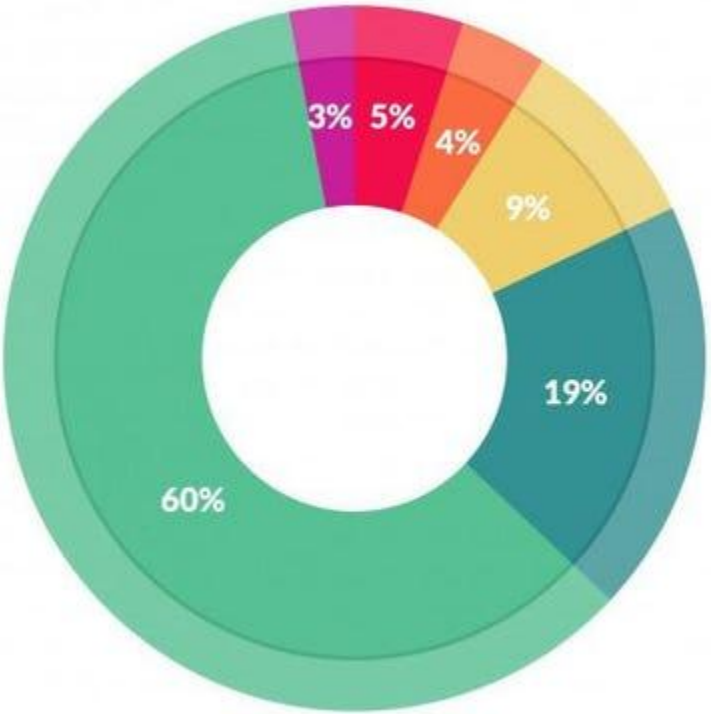
# AI 응용시스템의 이해와 구축

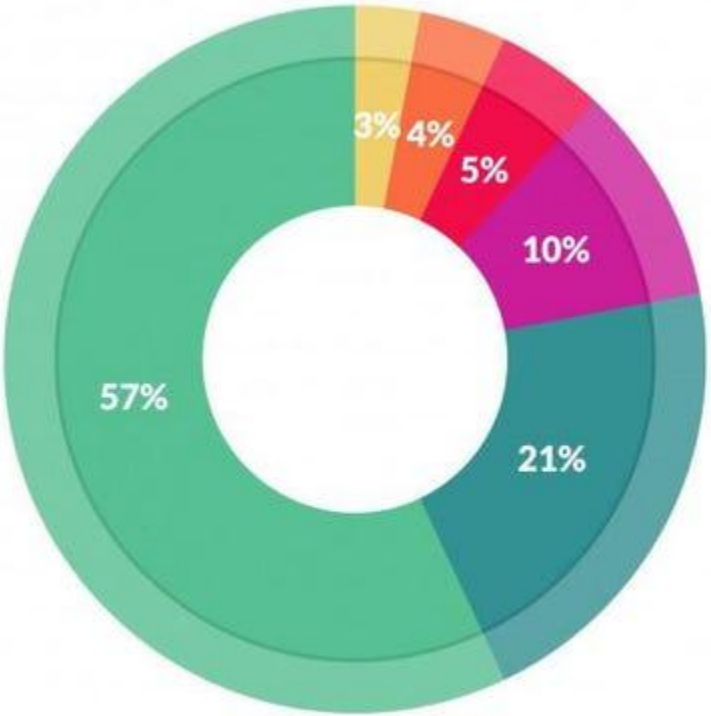
4강. Feature Engineering

# Feature Engineering




## What data scientists spend the most time doing

- *Building training sets: 3%*
- *Cleaning and organizing data: 60%*
- *Collecting data sets; 19%*
- *Mining data for patterns: 9%*
- *Refining algorithms: 4%*
- *Other: 5%*




What's the least enjoyable part of data science?

- *Building training sets: 10%*
- *Cleaning and organizing data: 57%*
- *Collecting data sets: 21%*
- *Mining data for patterns: 3%*
- *Refining algorithms: 4%*
- *Other: 5%*

*Cleaning Big Data: most time-consuming, least enjoyable data science task, Forbes*

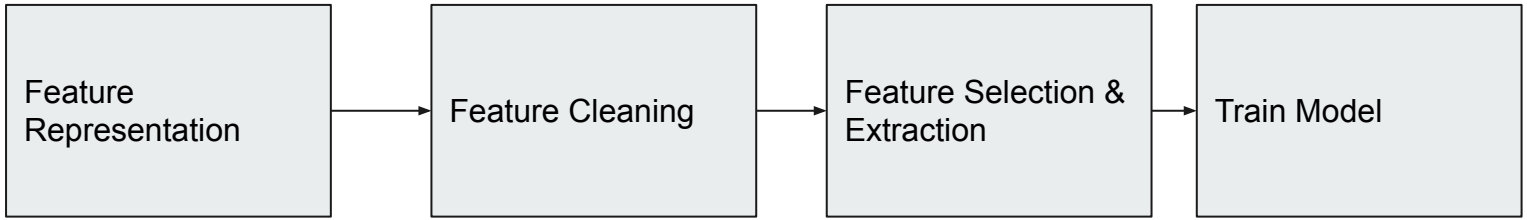
# Feature Engineering이란

- 모델 정확도를 높이기 위해 주어진 데이터(raw data)를 예측 모델이 잘 표현할 수 있는 feature로 변형하는 과정.
- 종종 ML알고리즘을 작동하기 위해 데이터의 **도메인 지식**을 활용하기도 한다.
- 모델링 알고리즘에 따라 Feature Representation을 적절하게 변형하기도 한다.

# Feature Engineering

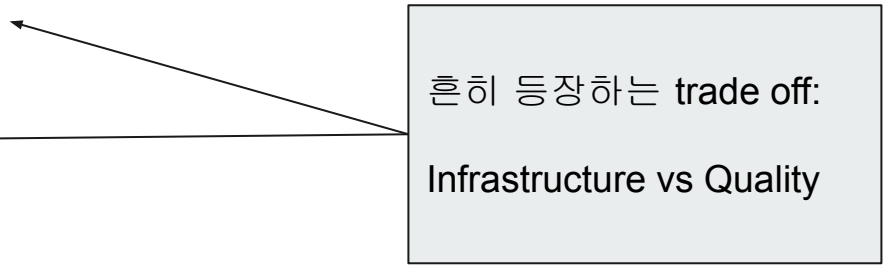
모델의 학습능력을 돕기 위하여 피쳐의 형식을 변경하거나 피쳐를 골라내는 작업이 필요함.

1. Feature Representation: 피쳐를 어떻게 **표현**할것인지.
2. Data Cleaning
3. Feature Selection
4. Feature Extraction



# Feature Engineering의 목표

- 머신러닝 알고리즘에 걸맞는 적절한 인풋 데이터셋을 준비
- 머신러닝 모델의 **성능 향상**
  - 모델 정확도, 추론 속도
- 모델 파이프라인의 **cost 저하**
  - 데이터셋의 크기



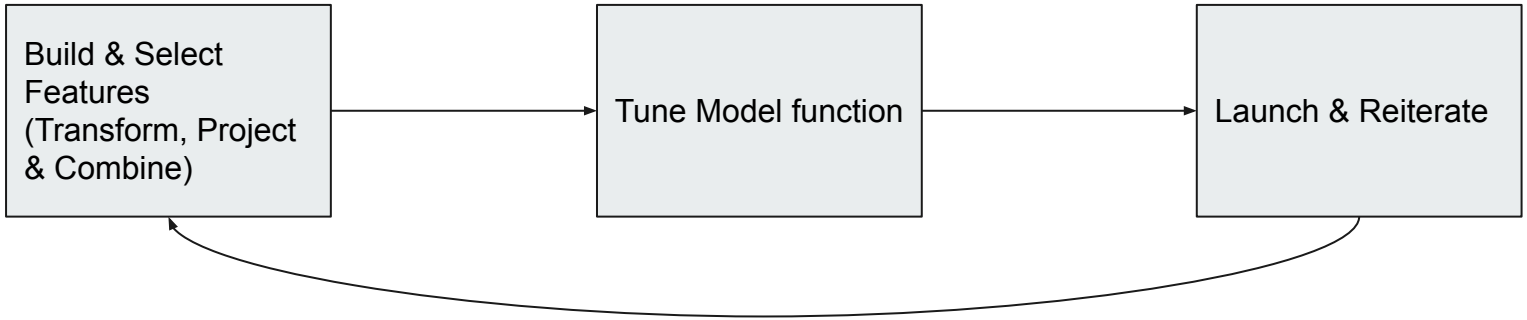
# Feature Engineering Pipeline

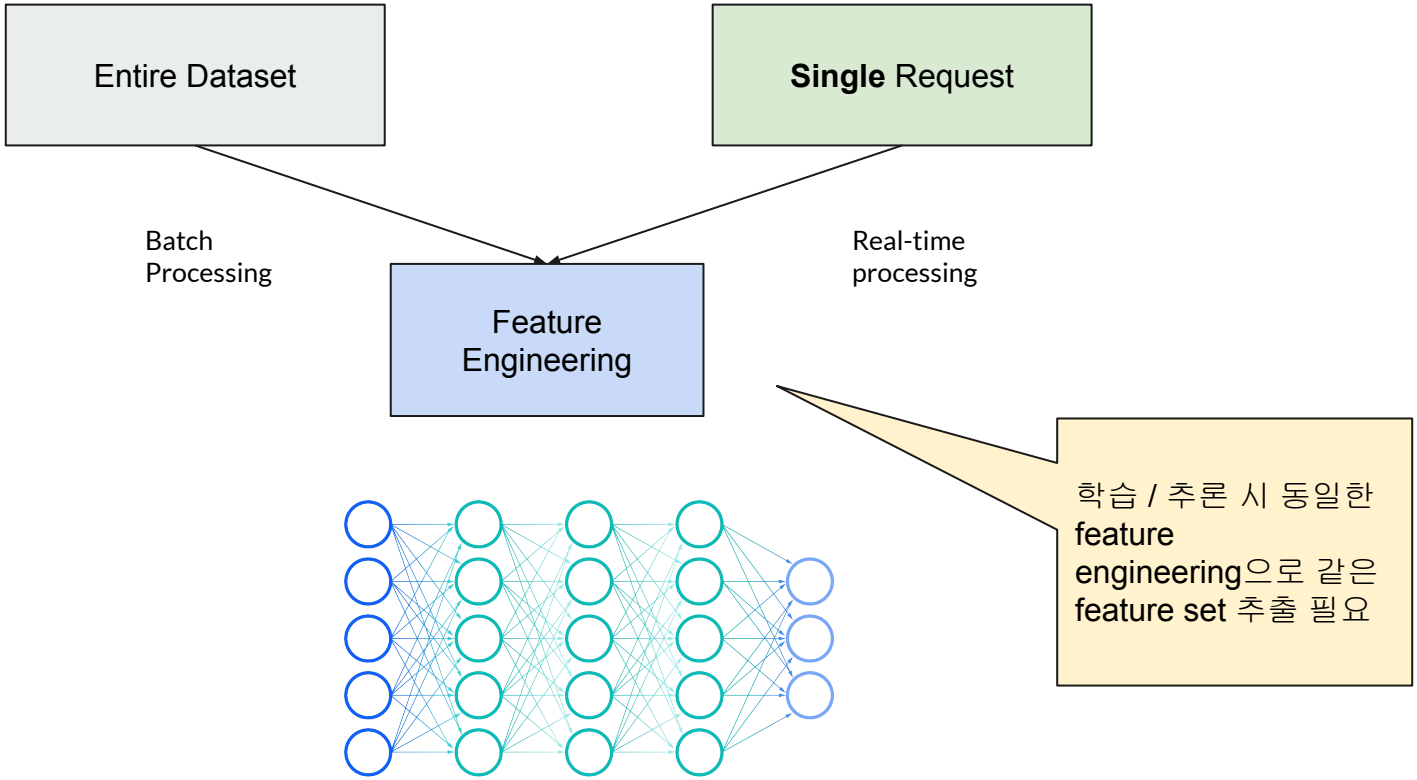
"피쳐엔지니어링 파이프라인 / 데이터 파이프라인"의 라이프사이클:

모델의 정확도와 리소스 활용을 반복적으로 발전하는 과정

- transforming: 데이터 클리닝
- projecting:
- combining features to a new version of data set

Transform: 데이터 클리닝
Project: 고차원의 데이터에서 피처 추출
Combine: 피처 추출/ 크로싱

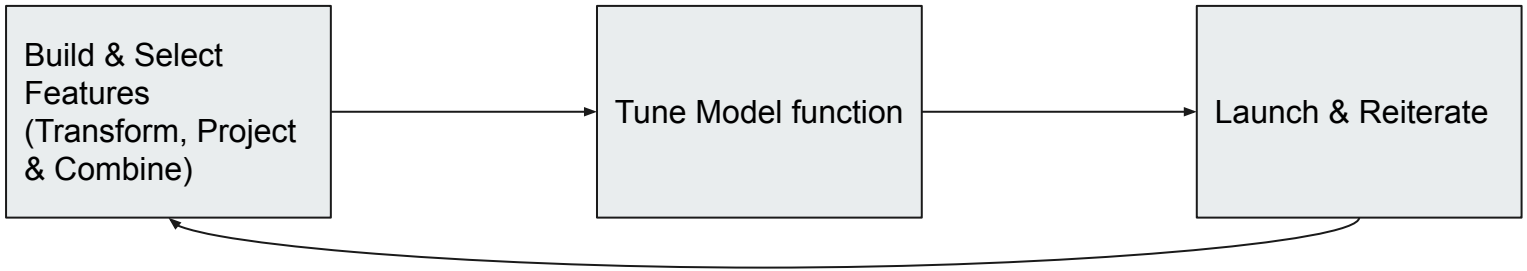



Entire Dataset
Single Request
Batch Processing
Real-time processing
Feature Engineering
학습 / 추론 시 동일한 feature engineering으로 같은 feature set 추출 필요

# Feature Engineering

최소한의 피쳐셋으로 모델을 구성하고, 차차 반복적으로 피쳐들을 넣음.

- 가장 predictive power가 강하다고 생각하는 피쳐들로 시작
- Compute 리소스 절약



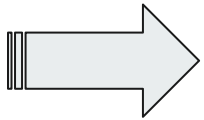
# Feature Representation

# Feature Representation

주어진 raw data를 피쳐벡터로 변형하는 작업에서 피쳐를 어떻게 표현할지를 결정.

**Raw Data**

```
patient 0:
        name: “Ken Adams”
        age: 65
        gender: male
        bp: 160, 80
        medical_history {
                diabetes,
        }
}
```



**Feature Vector**

```
[
        “Ken Adams”,
        65,
        0,
        160,
        80,
        [0, 0, …, 1, … 0]
]
```

# Feature Representation

피쳐 생성에서 필요한 작업

- 주어진 raw data를 피쳐로 맵핑
- 수치 데이터 (numeric features)를 어떻게 표현?
- 카테고리 데이터(categorical features)를 어떻게 표현?
- “도메인 지식"을 활용할수 있나?

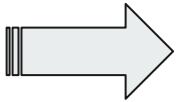
# Numerical Features

수치데이터는 인코딩 없이 그대로 맵핑이 가능
- 모델 weight에 feature value 그대로 연산 가능하기 때문.

특별한 인코딩 없이 바로 수치로 feature 저장.

**Raw Data**

patient 0:
age: 65
bp: 160, 80
weight: 64.5cm
}



**Feature Vector**

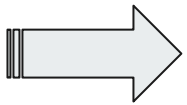
[
age: 65,
bp_systolic: 160,
bp_diastolic: 80,
weight: 64.5
]

# Categorical Features

Categorical Feature는? One-hot encoding으로 인코딩 가능.

Raw Data

patient 0:
    name: “Ken Adams”
    gender: male
    medical_history {
        **diabetes,**
    }
}



Feature vector

[
    “Ken Adams”,
    0,
    **[0, 0, …, 1, … 0]**
]

# One-Hot Encoding

- 단어를 표현하는 가장 기본적인 표현 방법.
- 단어집합(Vocabulary): 서로 다른 단어들의 집합. E.g. {apple, apples, …}
- Vocabulary에 있는 단어에 고유한 정수를 부여하여 인코딩.
- 표현하고 싶은 단어의 고유한 정수를 인덱스로 간주하고, 해당 위치에 1을 부여 (아니면 0).

“나는 국민대학교 소프트웨어 융합대학원 인공지능 전공 학생이다"

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 국민대학교 | 나는 | 소프트웨어 | 융합대학원 | 인공지능 | 전공 | 학생이다 |

# 단어 표현

“국민대학교"를 위 테이블로 표현하려면?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 |

“소프트웨어"?

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical

text = "나는 국민대학교 소프트웨어 융합대학원 인공지능 전공 학생이다"

tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
print('단어 집합 :',tokenizer.word_index)
```

```
단어 집합 : {'국민대학교': 1, '나는': 2, '소프트웨어': 3, '융합대학원': 4, '인공지능': 5,
'전공': 6, '학생이다': 7}
```

```
sub_text = "나는 국민대학교 학생이다"
encoded = tokenizer.texts_to_sequences([sub_text])[0]
print(encoded)
```

```
[2, 1, 7]
```

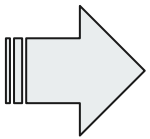
```
one_hot = to_categorical(encoded)
print(one_hot)
```

```
[[0. 0. 1. 0. 0. 0. 0. 0.] # 인덱스 2의 원-핫 벡터
 [0. 1. 0. 0. 0. 1. 0. 0.] # 인덱스 1의 원-핫 벡터
 [0. 0. 0. 0. 0. 0. 0. 1.]] # 인덱스 7의 원-핫 벡터
```

# Categorical Features

Raw Data

```
patient 0:
        name: “Ken Adams”
        gender: male
        medical_history {
                diabetes,
                hypertension,
                obesity,
        }
}
```



Feature vector

```
[
        “Ken Adams”,
        0,
        [0, 0, …, 1, … 0]
        [0, 1, … … … 0]
        [0, …, 1  … … 0]
]
```

**단점:**

- Vocab의 크기가 커질수록 필요 공간이 증가.
- 비슷한 단어들간의 상관관계 표현 어려움.

**필요공간:**

- Sparse Representation으로 해결

**상관관계:**

- Word Embedding 등으로 해결 (Word2vec, Bert)

# TensorFlow Code Example..

```
# From vocab list
vocab_feature_column = tf.feature_column.categorical_column_with_vocabulary_list(
                                        key=feature_name,
                                        vocabulary_list = [“diabetes”, “obesity”, … ])
# From vocab file
vocab_feature_column = tf.feature_column.categorical_column_with_vocaburarly_file(
                                        key=feature_name,
                                        vocabulary_file = “disease_names.txt”)
```

# Properties of “Good Features”

| | | |
|---|---|---|
| 충분히 존재하는 카테고리 피쳐 | Good: patient_gender,<br>**Bad: patient_id** | ID는 predictive power가 없음! |
| 명확히 이해하기 쉬운 피쳐 | Good: patient_age: 65<br>**Bad: patient_age: 1233444** | 디버깅할때 어려움. 차후 transformation 가능. |
| Magic value는 피할것 | Good: patient_age: 65<br>**Bad: patient_age: -1** | 누락값을 -1으로 표시. 다른 값과 상관관계 없음! |
| 시간이 지남에 따라 feature value가 변하지 않을 것 | Good: city_id: “kr/seoul”<br>**Bad: inferred_city_id: “2330”** | 2330이 다른 모델의 아웃풋. 모델 변경시 stability 저하 |

# Data Cleansing

# Data Cleaning

ML 엔지니어로서, 매우 많은 시간을 데이터 클리닝에 소요하게 된다..

- Scaling: 피쳐 값들을 스케일링 하기
- Outlier handling: 아웃라이어 데이터 포인트
- Binning: Continuous한 수치값을 Discrete한 빈(bin)으로 변화
- Scrubbing: 불필요한 데이터를 삭제
- Missing value handling: 누락 값 핸들링

# Feature Scaling

수치 데이터 (numerical feature)들의 레인지가 각각 서로 다를 때, 일반적으로 해당 numerical value 들을 표준화된 레인지로 스케일링하게 된다. 이를 통하여 얻는 장점은:

Gradient Descent이 더 빨리 converge하게 된다.

"NaN trap" 방지:
학습 알고리즘이 많은 operation을 할때 precision limit때문에 NaN가 되는 문제를 방지.

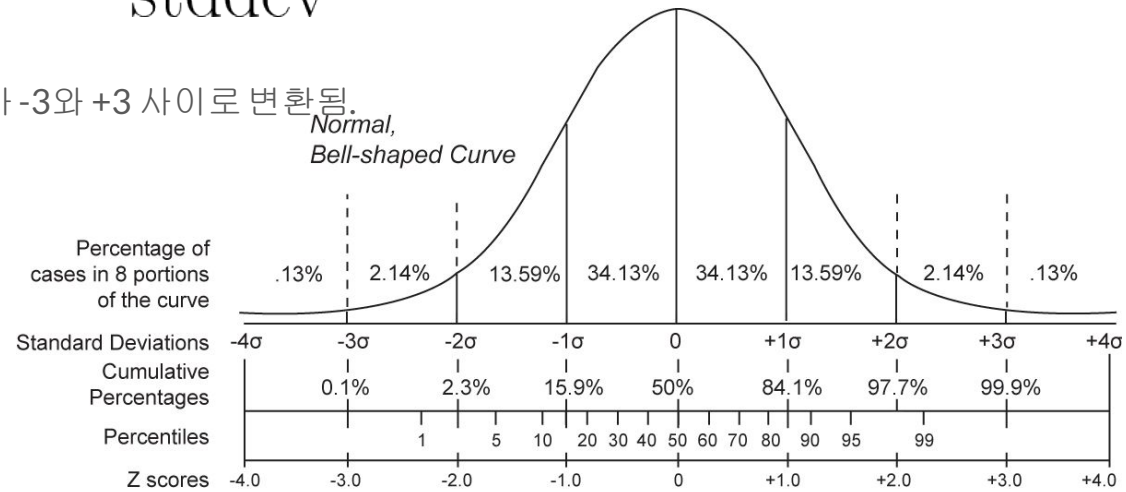
모델이 적절한 weight를 계산하는데 도움: feature scaling없이는 값이 큰 feature에 더 먼저 신경을 쓰게 됨.

# Feature Scaling: Z-score based scaling

**Z-score (표준 점수)**: 주어진 데이터 분포의 평균 (mean)값에서, 해당 수치가 얼마나 많은 standard deviation 의 거리만큼 떨어져 있는지의 수치.

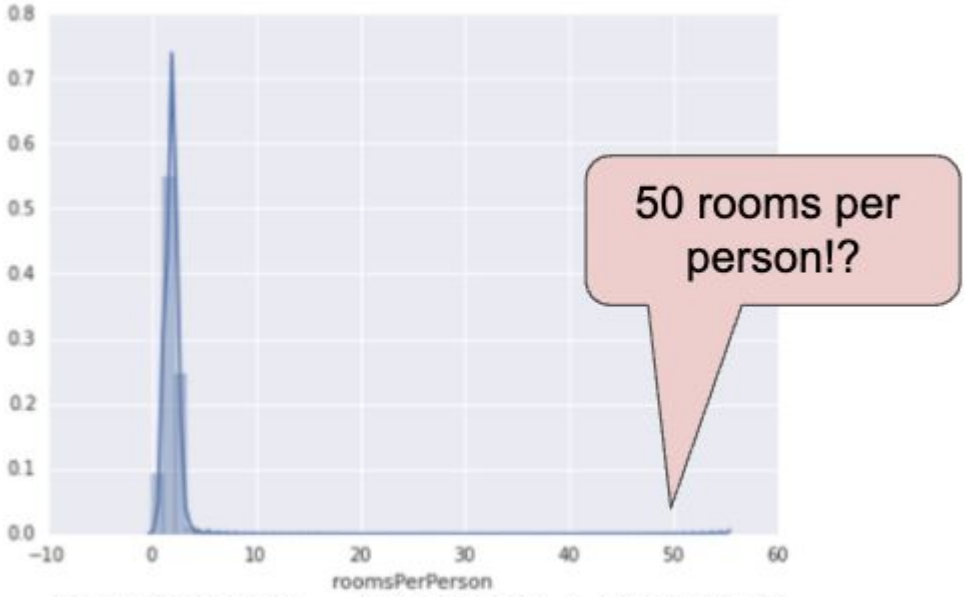
$$\text{scaled} = \frac{\text{value} - \text{mean}}{\text{stddev}}$$

Z-score로 스케일링할 때, 대부분의 수치가 -3와 +3 사이로 변환됨.



# Outlier Handling

(미국) 각 도시별 보유 주택 통계 데이터



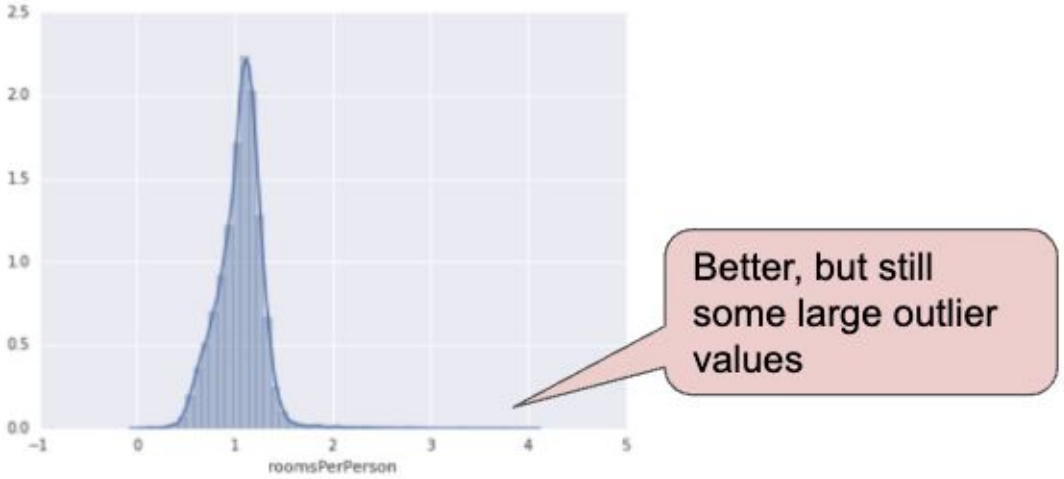

$$\text{roomsPerPerson} = \frac{\text{totalRooms}}{\text{population}}$$

인당 방 갯수가 너무 많은 outlier들이 등장.

# Outlier Handling: Apply Logarithm

각 수치마다 Log를 계산하여 분포를 좁힐 수 있음.

$$\text{roomsPerPerson} = log(\frac{\text{totalRooms}}{\text{population}} + 1)$$

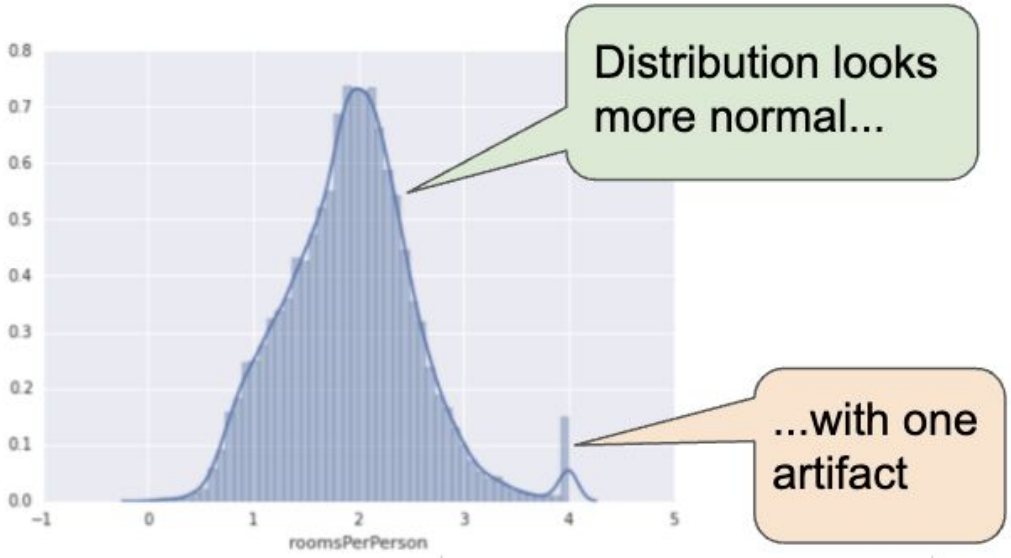



그래도 여전히 롱테일이 존재..

# Outlier Handling: Data Clipping

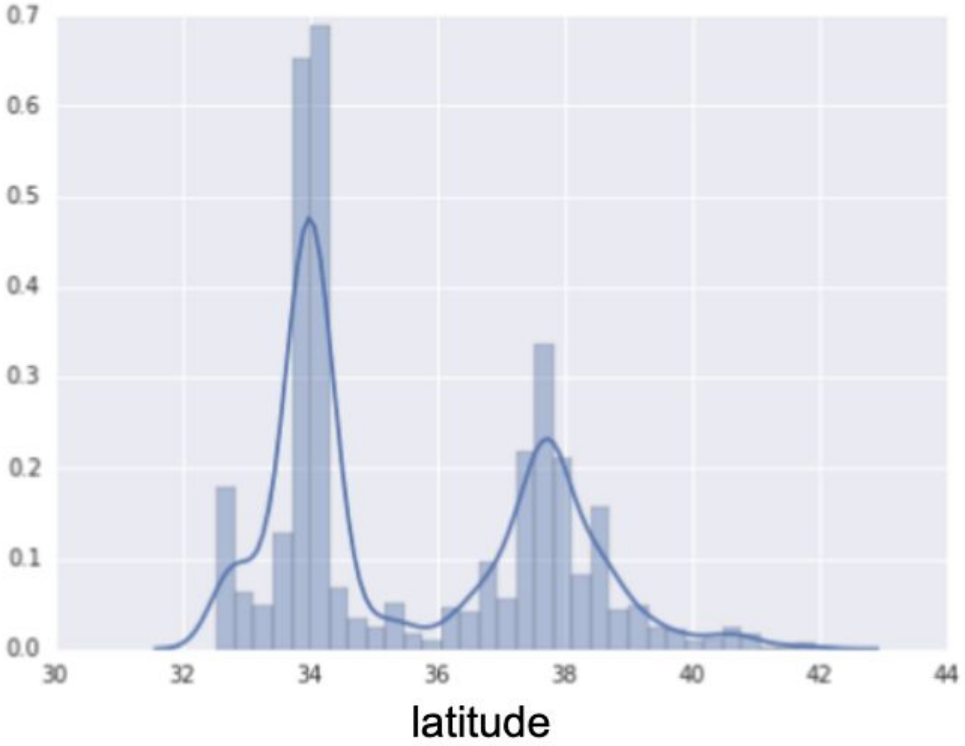
한사람당 사용하는 방의 갯수가 4개라고 가정하면..
나머지 데이터 포인트를 삭제

$$\text{roomsPerPerson} = \max(\frac{\text{totalRooms}}{\text{population}}, 4)$$




**Data Clipping**을 4로 실행했을 시,
분포가 더 정상적으로 보임.

# Feature Binning (Bucketing)




캘리포니아의 주택들을 위도(latitude)에 따라 분포하면 클러스터가 존재한다.

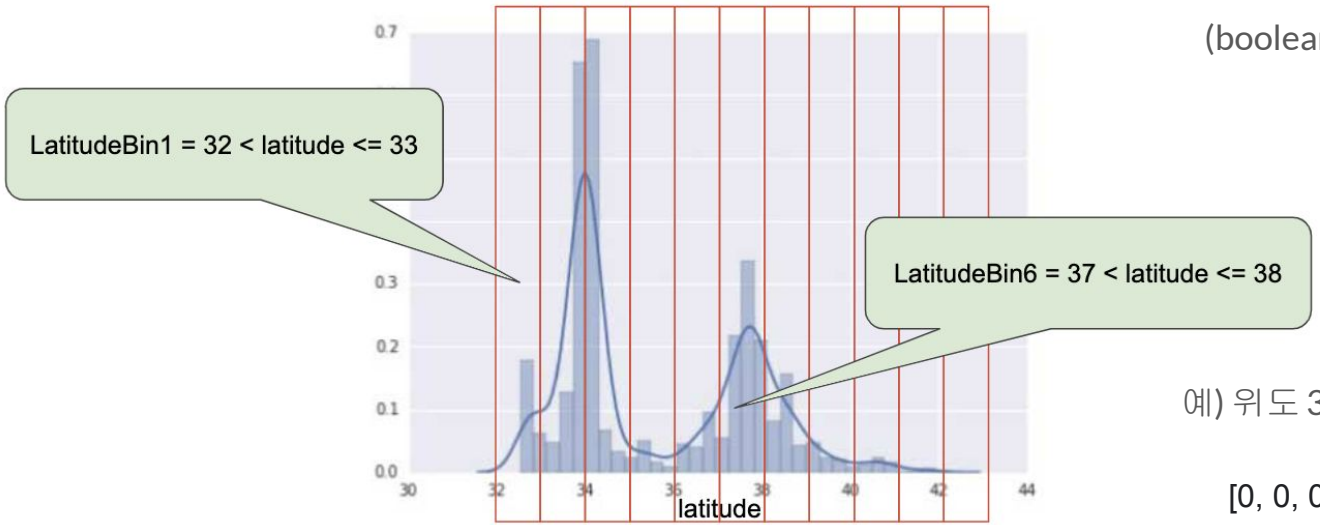
- 34: Los Angeles
- 38: San Francisco

주택 가격을 예측한다고 가정할 때, "위도"와 가격의 **선형 (linear)적인 상관관계**를 도출해내긴 어렵다.

# Feature Binning (Bucketing)

위도를 직접 feature로 사용하기보다, 위도 값을 11 개의 (bin) range로 나누어 카테고리 (boolean) 피쳐로 변경.

그리고 그 11 개의 피쳐를 One Hot Encoding으로 변경 가능.




예) 위도 37.4

[0, 0, 0, 0, 0, **1**, 0, 0, 0, 0, 0]

**Quantile Binning**: 실제 bin range를 나눌때는 동일한 간격으로 나누기보다, 각 Bin 마다 동일한 갯수의 데이터가 존재하도록 분열 ➔ 왜?

# Feature Scrubbing

실제 데이터에는 여러가지 이유로 믿을 수 없는 데이터가 존재한다.

- Omitted: 누락된 데이터
- Duplicate: 동일한 데이터가 여러번 기록됨.
- 레이블(Label) 오류: 데이터 수집 시 잘못 저장된 레이블 오류
- 데이터(Feature) 오류: 피쳐 데이터 수집 시 오류 (예: 특정 센서데이터 오류)

이러한 데이터 오류를 다양한 아래와 같은 Data Validation을 통해 모니터링하고, 잘못된 데이터를 삭제해야 한다.

- 최대, 최소값 (Maximum, Minimum)
- 평균, 중위값 (Mean, Median)
- 표준편차

# Break.

퀴즈 4

# Handling Missing Data

실제 데이터셋에 흔히 나오는 문제는 데이터 누락 (Missing Value)이다.

E.g. “[]”, “NaN”, “None”, ..

누락된 값이 많은 데이터로 학습하면 모델 정확도에 큰 영향을 미친다.

어떻게 핸들해야 할까?

# Handling Missing Data

1. Do Nothing

- 학습 알고리즘이 누락된 데이터를 처리하도록 허용.
- 일부 알고리즘은 누락데이터를 무시할 수 있는 기능이 있음 (예: xgboost, LightGBM)
- 일반적으로 누락데이터 처리를 못할 경우가 많음 (e.g. LinearRegression)

# Handling Missing Data

2. Remove Missing Value

- 누락된 데이터 자체를 제거.
- 누락된 데이터가 얼마나 많느냐 (반대로, 유의미한 데이터가 얼마나 많느냐)에 따라 모델 정확도에 큰 영향.

# Handling Missing Data: Imputation

**결측값 대체 (Imputation):**

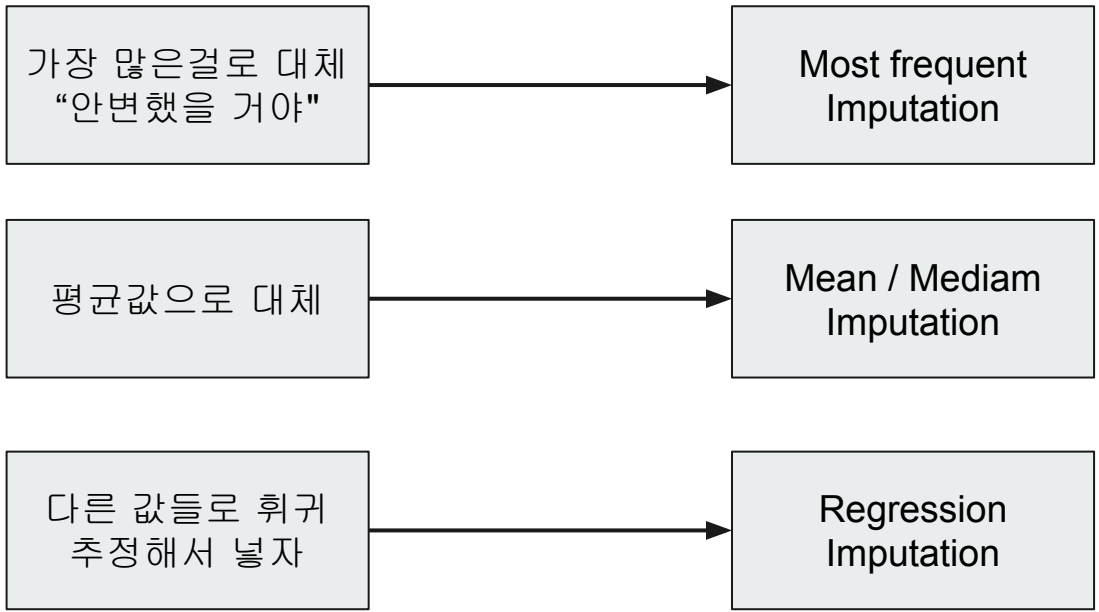
결측값 (missing value)이 존재할 때, 해당 결측값 대신 추정한 대체값을 계산하여 넣는 방법.

Imputation 방법은 크게 두가지로 나뉜다:

- Single Imputation
- Multiple Imputation

# Single Imputation

누락값(Missing value)들이 있으면 대체값을 어떻게 구할까?



# Most Frequent Imputation

## Most Frequent Value / Zero / Constant Imputation

- Most frequent: 가장 빈번히 나온 값으로 대체.
- Zero: 0으로 대체
- Constant: 지정한 상수값으로 대체

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | NaN | 41.0 | 6.984127 | 1.023810 | 322.0 |
| 1 | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 |
| 2 | NaN | 52.0 | 8.288136 | 1.073446 | 496.0 |
| 3 | 5.6431 | NaN | 5.817352 | NaN | 558.0 |
| 4 | NaN | 52.0 | 6.281853 | 1.081081 | 565.0 |
| 5 | 4.0368 | NaN | 4.761658 | 1.103627 | 413.0 |
| 6 | 3.6591 | 52.0 | 4.931907 | 0.951362 | 1094.0 |
| 7 | NaN | 52.0 | 4.797527 | 1.061824 | 1157.0 |
| 8 | NaN | 42.0 | 4.294118 | 1.117647 | 1206.0 |
| 9 | 3.6912 | 52.0 | 4.970588 | 0.990196 | 1551.0 |

most_frequent()

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 3.6591 | 41.0 | 6.984127 | 1.023810 | 322.0 |
| 1 | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 |
| 2 | 3.6591 | 52.0 | 8.288136 | 1.073446 | 496.0 |
| 3 | 5.6431 | 52.0 | 5.817352 | 0.951362 | 558.0 |
| 4 | 3.6591 | 52.0 | 6.281853 | 1.081081 | 565.0 |
| 5 | 4.0368 | 52.0 | 4.761658 | 1.103627 | 413.0 |
| 6 | 3.6591 | 52.0 | 4.931907 | 0.951362 | 1094.0 |
| 7 | 3.6591 | 52.0 | 4.797527 | 1.061824 | 1157.0 |
| 8 | 3.6591 | 42.0 | 4.294118 | 1.117647 | 1206.0 |
| 9 | 3.6912 | 52.0 | 4.970588 | 0.990196 | 1551.0 |

장점:

- 쉽고 빠르게 계산
- Categorical feature에 유용

단점:

- 다른 feature들과 무시
  - Invalid data 생성
- 데이터에 bias 생성

# Mean / Median Imputation

- 해당 feature column의 missing value들을, 다른 데이터셋의 평균(mean)이나 중위(median)으로 대체.
- 숫자형 데이터(Numerical feature)에만 사용 가능함.

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | NaN | 41.0 | 6.984127 | 1.023810 | 322.0 |
| 1 | 8.3014 | 21.0 | 6.238137 | 0.971880 | 2401.0 |
| 2 | NaN | 52.0 | 8.288136 | 1.073446 | 496.0 |
| 3 | 5.6431 | NaN | 5.817352 | NaN | 558.0 |
| 4 | NaN | 52.0 | 6.281853 | 1.081081 | 565.0 |
| 5 | 4.0368 | NaN | 4.761658 | 1.103627 | 413.0 |
| 6 | 3.6591 | 52.0 | 4.931907 | 0.951362 | 1094.0 |
| 7 | NaN | 52.0 | 4.797527 | 1.061824 | 1157.0 |
| 8 | NaN | 42.0 | 4.294118 | 1.117647 | 1206.0 |
| 9 | 3.6912 | 52.0 | 4.970588 | 0.990196 | 1551.0 |

mean()

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 5.06632 | 41.0 | 6.984127 | 1.023810 | 322.0 |
| 1 | 8.30140 | 21.0 | 6.238137 | 0.971880 | 2401.0 |
| 2 | 5.06632 | 52.0 | 8.288136 | 1.073446 | 496.0 |
| 3 | 5.64310 | 45.5 | 5.817352 | 1.041653 | 558.0 |
| 4 | 5.06632 | 52.0 | 6.281853 | 1.081081 | 565.0 |
| 5 | 4.03680 | 45.5 | 4.761658 | 1.103627 | 413.0 |
| 6 | 3.65910 | 52.0 | 4.931907 | 0.951362 | 1094.0 |
| 7 | 5.06632 | 52.0 | 4.797527 | 1.061824 | 1157.0 |
| 8 | 5.06632 | 42.0 | 4.294118 | 1.117647 | 1206.0 |
| 9 | 3.69120 | 52.0 | 4.970588 | 0.990196 | 1551.0 |

장점:

- 쉽고 빠르게 계산
- numerical feature에 유용

단점:

- 다른 feature들과 상관관계 무시
- 데이터에 bias 생성
- 동일한 값으로 모델 정확도에 영향 적음

# kNN Imputation: kNN

- missing value가 있는 데이터셋을, 근사한 다른 데이터셋들을 이용하여 추정값으로 대체.
- K개의 nearest neighbor를 찾은 후, 가장 많은 분류값으로 대체
- numerical feature 경우, NN중 거리에 따라 가중평균



장점:

- 다른 feature들과 상관관계를 이용하여 보다 정확
- numerical, categorical 사용 가능

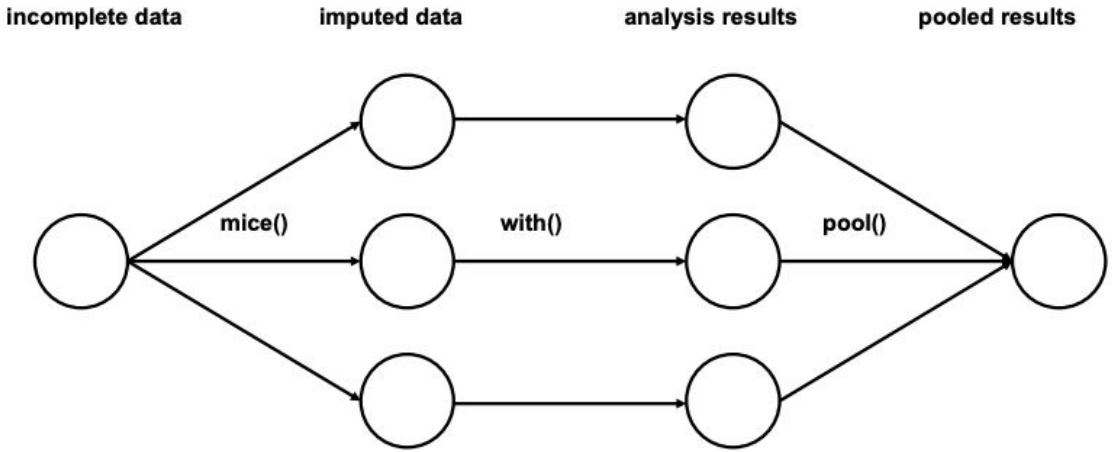
단점:

- Cost: 메모리 + 시간
- Outlier 민감

# Multiple Imputation

Single Imputation의 경우, 대체값으로 타 데이터셋과 비슷한 값으로 추정 대체

헌데, missing value라 하면, 원래 있던 값보다 신뢰도가 떨어져야 하지 않나?




1. 다양한 single imputation방법으로 대체값 생성
2. 해당 데이터셋으로 학습 + 결과 추론
3. 추론한 결과값을 Pooling 하여 사용

*When and how should multiple imputation be used for handling missing data in randomised clinical trials - a practical guide with flowcharts*, BMC Medical Research Methodology

*mice: Multivariate Imputation by Chained Equations in R*, Journal of Statistical Software

# Data Cleansing: Imputation Summary

- 모든 데이터셋에 완벽한 대체방법은 없음.
- 데이터 유형, 분포에 따라 성능의 차이가 나기도 한다.
- 한 feature당 누락된 데이터가 너무 많다면: Filter
- 혹은 계산하기 쉬운 방법으로 넣고 전체 모델링 시작.
    - 차후 다른 방법으로 하나씩 대체하거나 Multiple Imputation 방법을 사용

# Feature Selection

# Feature Selection

**Feature Selection**: 주어진 Feature들 중, 중요한 피쳐들만 골라내는 작업.

**피쳐의 중요도**: 수집된 데이터에서 특정 feature가 얼마나 중요한가?

- 얼마나 Output variable과 연관되었는가? Correlation Coefficient
- 얼마나 쓸모없는 데이터 인가? Variance Threshold

전체 피쳐들

중요한 피쳐만 필터

선택된 피쳐들

# Feature Selection

**VarianceThreshold**: 데이터셋에서 variance가 낮은 feature들을 threshold를 사용하여 필터할 수 있음.

e.g. zero-variance feature: 모든 데이터 샘플에 같은 값인 피쳐.

그럼 VarianceThreshold를 어떻게 세팅하나?

예) boolean 피쳐중 80% 이상 같은 값을 갖고 있는 피쳐들을 없애고 싶다.

boolean피쳐는 Bernoulli distribution이므로, Bernoulli random variable의 variance는

$$Var[X] = p(1-p)$$

# Variance Threshold Filter

$$Var[X] = p(1-p)$$
$$= 0.8 \cdot (1 - 0.8) = 0.16$$

```
>>> from sklearn.feature_selection import VarianceThreshold
>>> X = [[0, 0, 1], [0, 1, 0], [1, 0, 0], [0, 1, 1], [0, 1, 0], [0, 1, 1]]
>>> sel = VarianceThreshold(threshold=(.8 * (1 - .8)))
>>> sel.fit_transform(X)
array([[0, 1],
       [1, 0],
       [0, 0],
       [1, 1],
       [1, 0],
       [1, 1]])
```

첫번째 열이 필터링 됨.

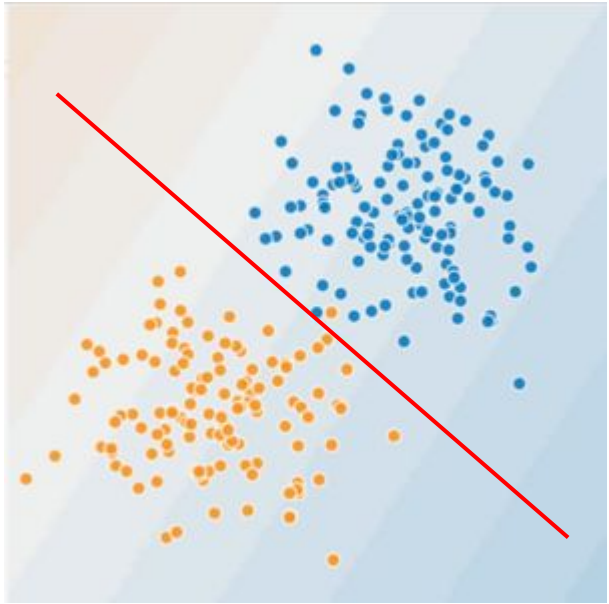
# Feature Extraction

# Feature Extraction

Feature Extraction에서는 주어진 데이터(raw data)에서 새로운 피쳐를 생성한다.

| Feature Crossing | 두개 이상의 feature를 “곱해서" 생성 | 비선형 (non-linear)적인 관계를 형성 |
| --- | --- | --- |

# Feature Crossing: Non-linearity



Task: 사진에서 병든나무 vs 건강한 나무를 분간하시오.

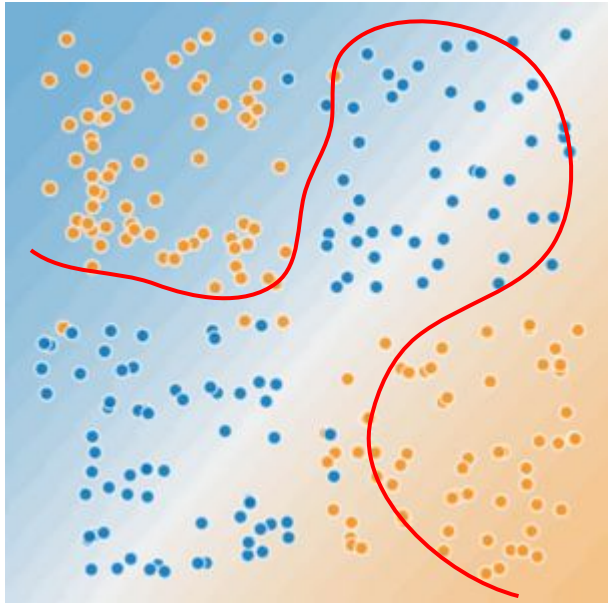
파랑: 병든 나무

노랑: 건강한 나무

| 선형적인 관계? | Yes | Feature Crossing 불필요 |
|---|---|---|

$$Y = f(x_1, x_2) = w_1 x_1 + w_2 x_2 + b$$

# Feature Crossing: Non-linearity



Task: 사진에서 병든나무 vs 건강한 나무를 분간하시오.

파랑: 병든 나무

노랑: 건강한 나무

| 선형적인 관계? | No | Feature Crossing 필요 |
| --- | --- | --- |

# Feature Crossing

- Feature crossing을 이용하여 피쳐들과 아웃풋과의 nonlinearity를 인코딩하려는 의도.
- 잇풋 벡터의 곱을 사용해서 새로운 피쳐로 사용.

$$x_3 = x_1 x_2$$

- 그리고 새로운 피쳐를 본래 linear classifier에 사용

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

# Feature Crossing

다양한 피쳐 크로스를 생성 가능

- [ A x B]: 2개의 피쳐를 크로싱
- [ A x B x C x D x E ]: 5개 피쳐를 크로싱
- [ A x A ]: 한개의 피쳐를 스스로 제곱하여 크로싱.

딥러닝에서는 Feature Crossing 필요 없다는데?

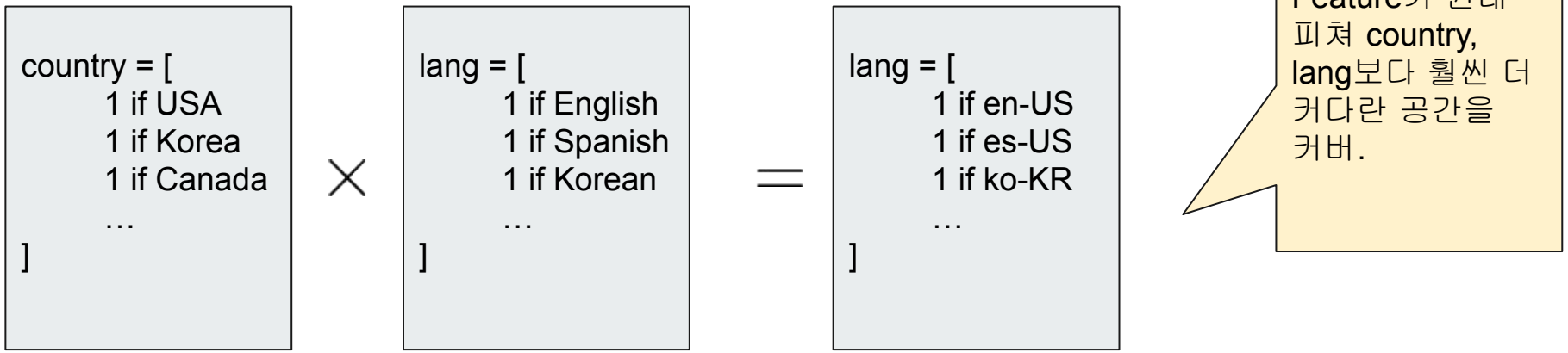
- Correct.
- Multilayer Perceptron (MLP)으로 feature crossing을 표현 가능.
- 그러나 어느 feature들을 어떻게 crossing할지를 미리 도와주는 단계.

# Feature Crossing: Categorical

일반적으로 카테고리 피쳐들을 자주 크로싱: country=USA, language=English

country = [0, 0, 0, 1, ….0]

lang = [ 0, 0, 0, … 1, …. 0 ]



# Other Feature Extraction

그 이외에도 각 문제의 도메인 지식을 사용하여 새로운 피쳐들을 생성하기도 함.

| | | |
|---|---|---|
| Text / Language | Stemming, 형태소 분리 | go, goes, gone<br>홍길동이 -> 홍길동, 이 |
| | n-grams | 홍길동이 학교에 갔다 ->“홍길동이 학교에”, “학교에 갔다" |
| | tf-idf | 문서의 단어 중요도 |
| Image / Vision | clipping | 이미지 잘라내기 |
| | channel | RGB, CMYK, … |
| | Canny filter | 엣지 디텍터 |

바로 옆에 등장하는 n개의 단어들과 **feature crossing**

# Feature Engineering in Deep Learning?

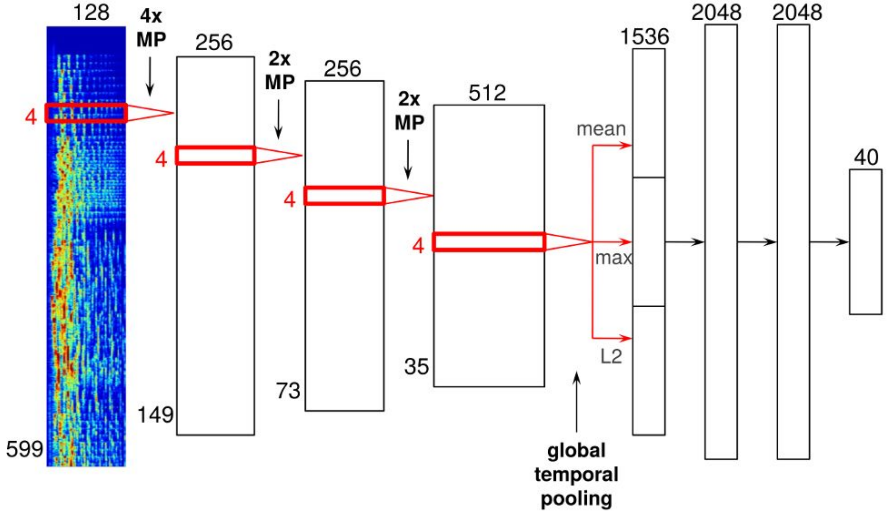
딥러닝에서는 raw input data에서 직접 학습을 통해 추론을 할 수 있다.

“From this perspective, a deep learning system is a fully trainable system beginning from raw input, for example image pixels, to the final output of recognized objects.”

그렇다면 딥러닝에선 Feature Engineering이 필요 없을까?

Yes and No.

# Spotify music recommendation model (link)




- 인풋: 푸리에변환에 의한 사운드 포맷
- Layer 1~3: Max pooling (scale)
- Layer 4: mean, max, L2 norm (time series)
- Layer 7~9: convolution
- 아웃풋: 아웃풋 통계 (convolution)

Layer 7~9을 제외한 나머지는 hard-coded 피쳐 변환.

모델 아키텍쳐 = Feature Engineering

# Steps for Feature Engineering

1. **도메인 지식이 있는가?** 그렇다면 손으로 피쳐를 작성하시오.
2. **선택한 피쳐들이 적합한가?** 아니라면 Normalize하시오
3. **피쳐들이 독립적인가?** 그렇다면 product feature / conjunctive feature들을 생성해보라 (리소스가 가능한 만큼)
4. **인풋 피쳐를 골라 내야 하나?** 아니라면 반대개념의 피쳐도 생성해보라
5. **개별 피쳐를 평가해볼 필요가 있는가?** 모델에 끼치는 영향 확인 혹은 전반적인 필터링이 필요?
6. **데이타가 "오염"되었다고 생각하나?** 아웃라이어들을걸러내시오
7. **어떤 모델로 시작해볼지?** 우선 linear predictor로 시작.

*An Introduction to Variables and Feature Selection*, Journal of Machine Learning Research (2003)

# Feature Engineering from Structured Data

Colab 튜토리얼

- (Colab의 영어버전) PetFinder 데이터: 애완견 입양 플랫폼, 입양 될지 안될지 예측.
- (한글버전): Cleveland Health 데이터셋.
- 정형데이터(structured data)를 사용한 feature engineering
  - Bin bucketing feature (버킷형 피쳐)
  - One Hot 인코딩
  - 임베딩 피쳐
  - 피쳐 크로싱